"Express Mail" mailing label number EV 164034225 US

Date of Deposit: January 27, 2004

Attorney Docket No.15397US01

SYSTEM, METHOD, AND APPARATUS FOR FIRMWARE CODE-COVERAGE IN COMPLEX SYSTEM ON CHIP

RELATED APPLICATIONS

[0001] [Not Applicable]

FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] [Not Applicable]

[MICROFICHE/COPYRIGHT REFERENCE]

[0003] [Not Applicable]

BACKGROUND OF THE INVENTION

[0004] One of the most difficult and time-consuming steps in the design of any system on chip (SOC) verification is firmware code coverage. The more complex the chip, the more difficult and time consuming the code-coverage task. Further adding to this, if the firmware is "embedded" within the SOC, then code coverage becomes even more complex and complicated.

[0005] Code coverage analysis includes finding areas of a program not exercised by a set of test cases, creating additional test cases to increase coverage, and determining a quantitative measure of code coverage, which is an indirect measure of quality. An optional aspect of code coverage analysis is identifying redundant test cases that do not increase coverage. Coverage analysis is critical

and important for complex system-on-chip verification and assures quality of set tests.

[0006] Embedded code-coverage has been done in simulations. As designs get bigger and users integrate more large blocks together to create huge SOCs, simulation time becomes unbearably long. Simulations can sometimes take in the order of several weeks. Some existing coverage analysis techniques access test program source code and often recompile it with a special command. This requires additional memory within the chip as the test cases are compiled along with "actual" firmware. This requires also more memory for this "virtual" code and drives up the cost of the SOC.

[0007] Further limitations and disadvantages of conventional and traditional approaches will become apparent to one of skill in the art, through comparison of such systems with embodiments presented in the remainder of the present application with references to the drawings.

BRIEF SUMMARY OF THE INVENTION

[0008] Presented herein is a system, method, and apparatus for firmware code-coverage in a complex system on chip.

[0009] In one embodiment, there is presented a circuit for analyzing code coverage of firmware by test inputs. The circuit comprises an input and a memory. The input receives an address from a code address bus. The memory stores recorded addresses from the code address bus. The memory comprises a plurality of memory locations, where each of the memory locations mapped to a particular one of a corresponding plurality of addresses associated with the firmware. The contents of the memory location associated with the address received from the code address bus are incremented responsive to receipt of the address.

[0010] In another embodiment, there is presented a method for analyzing code coverage. The method comprises receiving an address from a code address bus, the address being associated with an instruction in a system on chip; and incrementing a memory location mapped to the address associated with the instruction.

[0011] In another embodiment, there is presented a circuit for analyzing code coverage of firmware by test inputs. The circuit comprises an input, and a memory. The input receives an address from a code address bus. The memory is operably connected to the input, and stores recorded addresses from the code address bus. The memory comprises a plurality of memory locations, each of the memory locations mapped to a particular one of a corresponding plurality of addresses associated with the

firmware. The contents of the memory location associated with the address received from the code address bus are incremented responsive to receipt of the address.

[0012] These and other advantages and novel features of the present invention, as well as details of an illustrated embodiment thereof, will be more fully understood from the following description and drawings.

BRIEF DESCRIPTION OF SEVERAL VIEWS OF THE DRAWINGS

[0013] **FIGURE 1** is a block diagram of an exemplary system on chip whereon the present invention can be practiced;

[0014] **FIGURE 2** is a block diagram describing the testing of a system on chip in accordance with an embodiment of the present invention;

[0015] **FIGURE 3** is a block diagram describing an exemplary firmware code coverage address monitor in accordance with an embodiment of the present invention;

[0016] **FIGURE 4** is a flow diagram for analyzing code coverage in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0017] Referring now to **FIGURE 1**, there is illustrated a block diagram describing the testing of a system on chip in accordance with an embodiment of the present invention. The testing can either be post-fabrication or pre-fabrication. In the post-fabrication case, the system on chip (SOC) 105 can include any one of a variety of integrated circuits comprising an embedded processor 110 that executes instructions. The instructions are collectively referred to as firmware. In the pre-fabrication case, the system on chip 105 can be implemented or comprise an emulator or portion thereof, configured to emulate a design representative of any one of a variety of integrated circuits comprising an embedded processor 110.

[0018] The SOC 105 also comprises code memory space 115 for storing the firmware. When the processor executes the instructions from the firmware, the address of the instruction is provided to a code address bus 120. The code address bus 120 provides the address to the code memory space 115. Responsive thereto, the code memory space 115 provides the instruction stored at the address to the processor 110.

[0019] During testing, the SOC 105 is provided with test inputs. The operation of the SOC 105 for the test inputs can be evaluated to determine whether the SOC 105 operates properly, by evaluating the outputs. However, the test inputs do not always cause execution of each instruction in the firmware. Where the test inputs do not cause execution of each instruction in the firmware, it is possible that the firmware includes an erroneous instruction that did not

affect the outputs, because the inputs did not cause execution of the erroneous instruction. Accordingly, it is beneficial to provide test inputs that achieve high code coverage.

[0020] Referring now to **FIGURE 2**, there is illustrated a block diagram describing the testing of an SOC in accordance with an embodiment of the present invention. The code address bus 120 is connected to a code coverage address monitor module 205. The addresses from the SOC 105 represent the firmware code that is being fetched from the code memory space 115.

[0021] The code coverage address monitor 205 is a standalone module that is interfaced with the code address bus 120. The code coverage address monitor 205 captures the addresses from the code address bus 120 to be used for code coverage analysis. In one embodiment, the code coverage address monitor can be implemented as an appropriately configured emulator, or portion thereof.

[0022] A code coverage analyzer 210 analyzes the captured data. The code coverage analyzer 210 can comprise a computer system execution software that runs algorithms to determine the code coverage. Code coverage measures the degree that each instruction in the firmware is executed by the test inputs. Code coverage includes statement coverage, decision coverage, condition coverage, multiple condition coverage, condition/decision coverage, path coverage, function coverage, call coverage, data flow coverage, and object code branch coverage.

[0023] Statement coverage reports whether each executable statement is encountered. Statement coverage is

also known as "line coverage", "segment coverage", and "basic block coverage". Basic block coverage is similar to statement coverage except the unit of code measured is each sequence of non-branching statements.

[0024] Decision coverage reports whether Boolean expressions tested in control structures are evaluated to both true and false. Condition coverage reports the true or false outcome of each Boolean sub-expression, separated by logic-and and logical-or, if they occur. Condition coverage measures the sub-expressions independently of each other.

[0025] Multiple condition coverage reports whether every possible combination of Boolean sub-expressions occur. Condition/Decision coverage is a hybrid measure composed by the union of condition coverage and decision coverage. Path coverage reports whether each of the possible paths in each function have been followed. A path is a unique sequence of branches from the function entry to the exit.

[0026] Function coverage reports whether each function or procedure is invoked. Function coverage is useful during preliminary testing to assure at least some coverage in all areas of the software. Broad, shallow testing finds gross deficiencies in a test suite quickly.

[0027] Call coverage reports whether each function call is executed. The hypothesis is that faults commonly occur in interfaces between software modules. Call coverage is also known as call pair coverage.

[0028] Data flow coverage is a variation of path coverage that considers the sub-paths from variable assignments to subsequent references of the variables. The advantage of this measure is the paths have direct

relevance to the way the program handles data. This measure generally does not include decision coverage.

[0029] Object code branch coverage reports whether each machine language conditional branch instruction both took the branch or fell through. This gives results that depend on the compiler rather than on the program structure since compiler code generation and optimization techniques can create object code that bears little similarity to the original source code structure.

[0030] Loop coverage reports how many times each loop body is executed consecutively. For "do-while" loops, loop coverage reports whether the body executed once or, if more than once, how many times.

[0031] Referring now to **FIGURE 3**, there is illustrated a block diagram describing an exemplary code coverage monitor 205. The code coverage monitor 205 comprises an address multiplexer (MUX) 305, an address counter 310, a non-volatile memory 315, and a data multiplexer (MUX) 320. The address MUX 305 selects either the code address from the SOC 105, or the address from the address counter 310. The data MUX 320 selects either a clear signal or an increment signal.

[0032] The non-volatile memory 315 is mapped to the addresses storing the instruction in the firmware in the code memory space 115 of the SOC 105. During operation, the non-volatile memory 315 receives an address from the address MUX 305, and data from the data MUX 320. Responsive thereto, the data received from the data MUX 320 is written to a memory location in the non-volatile memory 315 mapped to the address received from address MUX 305.

[0033] The code coverage monitor 205 can operate in one of two modes - a memory initialization mode and a monitoring mode. In the memory initialization mode, the code coverage monitor 205 clears the memory locations in the non-volatile memory 315. The address MUX 305 selects the address from address counter 310, while the data MUX 320 selects the clear data signal.

[0034] The address counter 310 is set to the first address in the non-volatile memory 315. When the address MUX 305 provides the address from the address counter 310 to the non-volatile memory 315 and the data MUX 320 provides the clear data signal, the memory location in the non-volatile memory 315 that is mapped to the address from the address counter 310 is cleared. Additionally, the address counter 310 increments to the next address. The foregoing is repeated until all of the locations in the non-volatile memory 315 are cleared.

[0035] In the monitoring mode, the address MUX 305 selects the code address received from the SOC, while the data MUX 320 selects an increment signal. When the address MUX 305 provides the address received from the SOC to the non-volatile memory 315 and the data MUX 320 provides the increment signal to the non-volatile memory 315, the contents stored in the memory location in the non-volatile memory 315 mapped to the address received from the SOC are incremented.

[0036] Referring now to **FIGURE 4**, there is illustrated a flow diagram for analyzing code coverage in accordance with an embodiment of the present invention. At 405, the address counter 310 is set to the first address in the non-volatile memory 315. At 410, the address MUX 305 is set to select

the address counter. At 415, the data MUX 320 is set to select the clear signal. When the address MUX 305 provides the address from the address counter 310 to the non-volatile memory 315 and the data MUX 320 provides the clear data signal, the memory location in the non-volatile memory 315 that is mapped to the address from the address counter 310 is cleared (420). Additionally, the address counter 310 increments (425) to the next address. The foregoing is repeated (430) until all of the locations in the non-volatile memory 315 are cleared.

[0037] The code coverage analyzer 305 then enters the monitoring mode. At 435, the address MUX 305 selects the code address received from the SOC, while the data MUX 320 selects (440) an increment signal. When the address MUX 305 provides the address received from the SOC to the non-volatile memory 315 and the data MUX 320 provides the increment signal to the non-volatile memory 315, the contents stored in the memory location in the non-volatile memory 315 mapped to the address received from the SOC are incremented (445).

[0038] The foregoing represents techniques for performing firmware code coverage analysis of firmware that is embedded in an SOC. Code coverage analysis is advantageously performed in a manner that does not require actual access to the program source code, does not require recompiling, and analyzes the code-coverage in real-time. Additionally, in the pre-fabrication case, verification of 100% code coverage can save chip data memory.

[0039] One embodiment of the present invention may be implemented as a board level product, as a single chip, application specific integrated circuit (ASIC), or with

varying levels integrated on a single chip with other portions of the system as separate components. The degree of integration of the system will primarily be determined by speed and cost considerations. Because of the sophisticated nature of modern processors, it is possible to utilize a commercially available processor, which may be implemented external to an ASIC implementation of the present system. Alternatively, if the processor is available as an ASIC core or logic block, then the commercially available processor can be implemented as part of an ASIC device with various functions implemented as firmware.

[0040] While the invention has been described with reference to certain embodiments, it will be understood by those skilled in the art that various changes may be made and equivalents may be substituted without departing from the scope of the invention. In addition, many modifications may be made to adapt particular situation or material to the teachings of the invention without departing from its scope. Therefore, it is intended that the invention not be limited to the particular embodiment(s) disclosed, but that the invention will include all embodiments falling within the scope of the appended claims.